On Dynamic Polymorphing of a Superscalar Core for Improving Energy Efficiency

Sudarshan Srinivasan, Rance Rodrigues, Arunachalam Annamalai, Israel Koren, *Fellow, IEEE*,
Sandip Kundu, *Fellow, IEEE*
Department of Electrical and Computer Engineering
University of Massachusetts at Amherst, MA, USA

***Abstract*—The computational needs of a program change over time. Sometimes a program exhibits low instruction level parallelism (ILP), while at other times the inherent ILP may be higher; sometimes a program stalls due to a large number of cache misses, while at other times it may exhibit high cache throughput. Asymmetric Multicore Processors (AMP) have been proposed to allow matching the computing needs of a thread to a core where it executes most efficiently. Some of the recent works focus on AMPs consisting of a monolithic large out-of-order (OOO) core and a small in-order (InO) core. Dynamic swapping of threads between these cores is then facilitated to improve energy efficiency of the threads without impacting performance too negatively. Swapping decisions are made at coarse grain instruction granularities to mitigate the impact of migration overhead. This excludes many opportunities for swap at a fine granular level. In this paper we consider a single superscalar OOO core that can morph itself dynamically into an InO core at runtime. In order to determine when to morph from OOO to InO and vice-versa, we rely on certain hardware performance monitors. Using these performance monitors we estimate the energy-delay-squared product (ED^2P) for both modes of operation, which is then used to make morphing decisions. The morphing hardware support is simple and is already available in certain Intel processors to facilitate debug. The proposed scheme has low migration overhead, that enables fine-grain morphing to achieve more energy efficient computing by trading a small loss of performance for much greater energy reduction.**

Keywords*—*Core Morphing; Asymmetric Multicore Processor (AMP); Out-of-Order (OOO); In-Order (InO); Performance Monitoring Counter (PMC)

I. Introduction

Advancements in technology have resulted in increased circuit performance and the ability to pack more transistors into a small area. The higher device density and rising frequency led, unfortunately, to a power density problem which paved the way for the multicore era [1]. In current processor ICs, a single and very powerful processor has been replaced by many symmetric cores (Symmetric Multicore Processor (SMP)), each with more modest computational capabilities.

Symmetric multicore processors are better suited for Thread Level Parallelism (TLP), and thus, the performance suffers whenever sequential applications with high instruction level parallelism (ILP) are encountered [2]. Furthermore, it is known that different workloads require different computational resources to maximize performance/power. Even during the execution of a given workload, resource requirements may vary with time due to changes in program phases [3]. Asymmetric Multicore Processors (AMP) with the capability to cater to the diverse needs of workloads were introduced as a potential solution to this conundrum [4]. Often, the explored AMPs employ two kinds of cores: out-of-order (OOO) big cores and in-order (InO) small cores. The big cores provide higher performance while the in-order small cores are more power efficient. As the benefits of such AMPs are highly dependent on a proper thread-to-core assignment, the threads are swapped between the cores at runtime so that the objective function (e.g., performance, performance/power, or energy) is improved for the current program phase.

Thread swapping, however, incurs non-negligible costs. The swapping overhead can vary from a few thousand to millions of cycles [5] depending on the algorithm employed to swap threads and the mechanism to exchange contexts. To amortize the large overhead associated with thread swapping, in most proposals, thread swapping decisions are made at the granularity of hundreds of thousands to millions of instructions [5]. Unfortunately, numerous opportunities to improve performance/power and/or energy-delay-squared product (ED^2P) at a more fine grained instruction granularity are missed by such approaches [6]. Therefore, there is need for a mechanism to take advantage of such opportunities without incurring large thread swapping penalties.

In this paper we propose a novel core morphing mechanism that reaps most of the benefits of AMPs, without incurring the high penalty associated with thread swapping. Our proposed mechanism introduces heterogeneity within the same core by morphing it from OOO to InO core and vice-versa. Certain Intel processors feature a special debug mode in which the OOO core turns into an InO core [7]. We extend the use of this mechanism for improving energy efficiency by opportunistically switching to the InO mode, if deemed beneficial. As the morphing is performed within the same core and the architectural states are retained, the overheads associated with our scheme is negligible.

At a base level, we consider a single complex superscalar core that operates in the OOO mode providing high performance. However, during low IPC phases of the program, the operation mode may be switched to the InO mode for energy reduction. A similar switch is made from InO to OOO when these benefits are predicted to have diminished. To achieve energy benefits without impacting performance significantly, we use the energy-delay-squared product (ED^2P) as our optimization metric. The central idea of our proposal is the online estimation of the expected ED^2P of the executing thread in the other mode, while it is being executed in the

978-1-4799-2987-0/13/$31.00 ©2013 IEEE

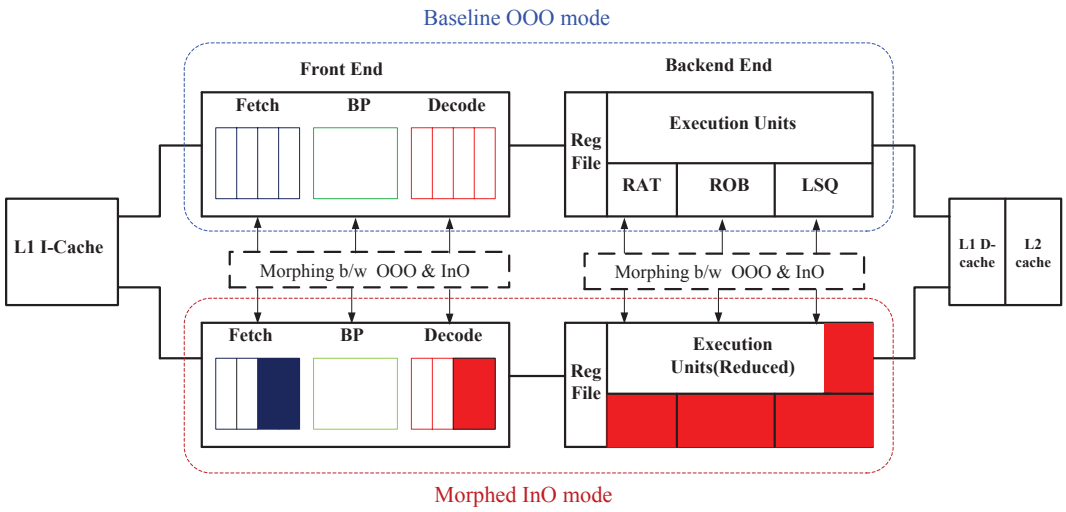

Fig. 1. High-level view of the proposed core morphing scheme. The baseline OOO mode is shown at the top. The shaded regions indicate the units of the baseline core that are power-gated to facilitate in-order execution in InO mode.

current mode. The estimation is made possible by employing the performance monitoring counters (PMCs) of the baseline core.

Since the proposed scheme makes use of existing facilities in a processor, it incurs no hardware overheads unlike several comparable schemes [6], [8], [9], [10]. The key contributions of this paper are:

1) Dynamic morphing within the same core between OOO and InO modes using existing debug mechanisms in current microprocessors.
2) Online estimation scheme to estimate power and performance on the same core as well as on other cores which help us to make best thread to core assignment.

II. Related work

There have been a number of studies on dynamic morphing of multicores processors for reducing power or improving energy efficiency. Kim *et. al.* and Tarjan *et. al.* proposed morphing schemes that fuse multiple simple cores together into a large OOO core on demand [8], [9]. These approaches suffer from additional latencies in the pipeline that arise from combining resources from various cores. Khubaib *et al.* described a morphing scheme where a baseline OOO core morphs itself into a Simultaneously Multithreaded InO core depending on the number of incoming threads [10]. These schemes require significant changes to the designed microarchitecture. Lukefahr *et al.* presented a morphing scheme where heterogeneity is introduced into the same core by provisioning two execution backends for the same frontend [6]. One backend is an OOO while the other is InO. Both backends share the cache and fetch units. Our scheme differs from Lukefahr *et al.* in that we have a common execution backend while they have two. Apart from larger area, their scheme requires the architectural states to be transferred from one backend to the other, which creates a larger migration overhead. In our scheme, morphing between InO and OOO can occur at much finer granularity. To take advantage of this opportunity, we need to assess and predict the benefits of morphing *dynamically*. In this paper, we present an estimation technique for assessing whether the program will be better-off running in the other mode based on statistics collected using certain hardware performance counters. This is the main thrust of this paper.

III. Proposed Approach

In this section, we describe both the architectural and implementation details of the proposed core morphing scheme that supports switching between OOO and InO modes at fine-grained time intervals.

A. Architectural Details

Figure 1 shows the considered baseline core which is a 4-way issue OOO superscalar core. The backend of the baseline core includes a register alias table (RAT), load/store queue (LSQ) and Re-Order Buffer (ROB) to facilitate OOO execution and in-order commit. During high-ILP program phases, significant performance benefits are achieved by executing the thread on the OOO baseline core. However, when the processor is waiting for long-latency memory operations to complete or stalls due to dependencies, most of the core resources are idle wasting static power.

For such low-IPC phases, a low-power InO core may be more energy efficient. In order to identify the difference between the power consumption in the OOO and InO modes of operation, we analyzed the various components of the power spent in each mode of operation. As expected, the OOO mode consumes considerably more power than the InO mode. The OOO mode relies heavily on speculative execution by making use of data structures such as the ROB and reservation stations to ensure OOO execution but in-order commit. Data movements between these structures is a major contributor to power consumption. For some phases of a program, this increase may not be commensurate with the performance benefits resulting in poor energy efficiency. We also see that in the issue and execution stage, power for the OOO mode are significantly higher than in the InO mode. These are the stages where the data structures are used and accessed the most. When such an increase in power is not accompanied with a significant performance gain, a switch in mode from OOO to InO may be beneficial. To this end, during low-ILP/memory intensive phases, we power off the ROB, RAT,

and LSQ, enabling only in-order execution/commit. Thus, the baseline OOO core is opportunistically morphed into an InO core providing significant power benefits. As the performance of the core in InO mode is expected to be low, we reduce the fetch width of the core from 4 to 2, and further, power off half of the decoders and shut-down few of the multiple execution units. The InO mode (see Figure 1) is thus more power efficient than the baseline OOO mode. While in InO mode, if the program moves to a high-ILP phase, the shut down units are powered on, reverting back to the baseline OOO execution. The architecture for this morphing is considerably simpler than that presented in [6]. Certain Intel processors are already reported to have support for switching from OOO to InO mode [7].

Morphing from the OOO to InO modes of operation needs to be done at runtime. This requires a mechanism that makes dynamic decisions depending on the characteristics of the currently executing workload. A description of the proposed mechanism is presented next.

B. Implementation Details

Prior knowledge about the computational resource requirements of different applications is generally not available beforehand. Hence, there is a need for an online mechanism to characterize the time-varying program behavior and determine the appropriate mode (OOO or InO) at runtime such that the ED^2P of the executing application is minimized.

The current characteristics of the application being executed on a core can reveal considerable information about how suitable the core is to that application. For example, an application phase that results in a significant number of misses in the level-1 cache will result in low performance and high energy consumption. Executing this phase on an InO core would make more sense with respect to energy usage. The optimization metric we target is energy-delay-squared-product (ED^2P). This metric gives higher priority to performance over energy while a simple EDP metric would favor InO nearly always, costing in performance for a dynamic morphing scheme. In order to assess the current characteristics of the application being executed, we make use of Performance Monitoring Counters (PMC).

To estimate the ED^2P, both performance and energy (power) need to be measured or estimated. Performance measurement is straightforward, while real-time power or energy measurement is not. PMCs have been used as a proxy to estimate power in the past [11], [12] and we follow a similar approach. Note that most previous work makes use of PMCs to estimate power on the same core while we need to estimate power and performance on the currently active mode (OOO/InO), as well as the other mode (InO/OOO) to make an informed decision.

a) PMCs explored in this study: There are many events that take place in a modern processor but some of them provide better hints than others about the performance and power of the currently executing application. To this end, we have explored fourteen different performance counters. We considered (i) the number of retired instructions of each type (integer, floating-point etc.), (ii) memory hit and miss counters (level-1, level-2 and TLB), (iii) number of mis-predicted and correctly predicted branch instructions, (iv) number of instructions fetched and instructions retired per cycle (IPC), and (v) pipeline stalls which consist of stalls resulting due to lack of reservation stations, load/store queue, RAT and ROB slots.

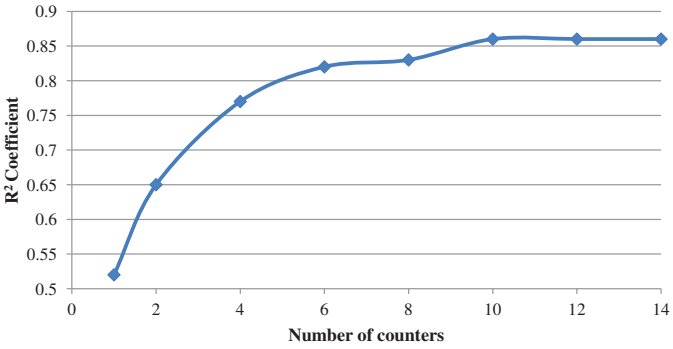

Fig. 2. Variation in the R^2 coefficient while estimating the performance in InO mode using the values of PMCs observed in the OOO mode.

b) Shortlisting the PMCs: In general, we expect a higher estimation accuracy using a large number of counters. However, there is a limit on the number of counters that may be accessed at the same time. This limit varies from one architecture to another. For example, in the Intel XScale processor [11], only two counters may be accessed while for the AMD Phenom processor, at most five counters may be accessed at the same time [12]. There is, therefore, a need to find a minimal subset of PMCs that have the most impact on power and performance both in the currently active mode, and the other mode.

To accomplish the task of making the right choice of PMCs, we devised an efficient heuristic that searches the counter space iteratively. During each iteration, our counter selection algorithm picks a new counter that best fits the estimating parameter (performance or power) along with the set of counters already chosen in previous iterations. We tried only linear models for curve-fitting and the best fit is qualified by the R^2 correlation coefficient. During the initial few iterations, the value of R^2 increases steeply as more counters are added, but it tends to saturate later. The best set of counters is around the region where the R^2 coefficient is saturating.

The result of one such counter selection experiment is shown in Figure 2. Here, the expected performance of the application in InO mode is estimated using the values of PMCs observed in the OOO mode. As expected, increasing the number of counters yields higher R^2 values. However, we arrive to the point of diminishing returns after 6 counters. These 6 counters were IPC, number of retired load and store instructions, pipeline stalls, branch mis-predictions and level-1 cache hit rate. Similar experiments were run to obtain expressions that can be used to estimate both performance and power in the two modes using PMCs.

The average error observed when using PMCs in one mode (OOO/InO) to predict the power in that mode as well as the performance and power in the other mode (InO/OOO) is shown in figure 3. While estimating the OOO parameters (IPC and power) from the InO mode using PMCs in the InO mode, the average error in estimating IPC and power is around 16% and 10%, respectively. Similarly, the average error in computing the InO parameters from OOO mode was found to be 15%

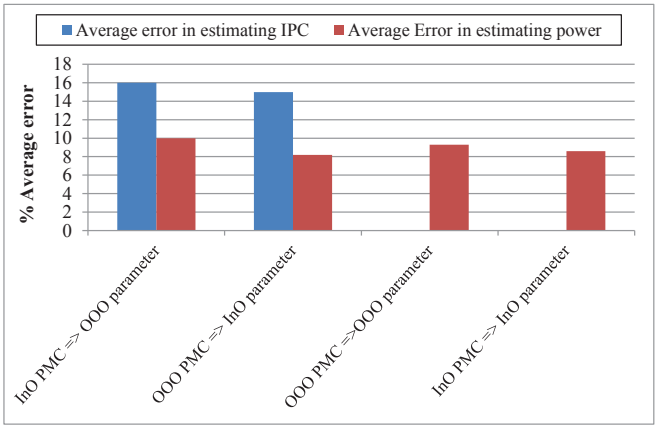

Fig. 3. % Average error observed in estimating IPC and power of OOO (InO) mode using InO (OOO) counters.

and 8%, respectively.

C. Dynamic Switching between OOO and In0 core

We have seen how ED^2P can be estimated using PMCs. A decision to move to the alternate mode of operation is made if it is expected to provide a better (lower) ED^2P. The decision to switch the mode of operation should be one of high confidence. Otherwise, we risk running into oscillations between the two modes. This will likely negate all benefits of the proposed scheme. Hence, it is necessary to ensure that the decision to change operation mode is made only if the resulting benefit is expected to be long term. Thus, we plan to rely on the estimated ED^2P in our future work as explained below.

After a certain number of retired instructions, referred to as window, a tentative morphing decision about the best mode (OOO or InO) could be made based on the estimated ED^2P. To avoid too frequent switching between the modes (InO and OOO), we plan to wait until the new execution phase is stabilized. To that end, we base our morphing decision on the most frequent tentative decision made for the past n retired instructions. Morphing overheads for our scheme is estimated to be low as there is no need to change the state of the register file, caches and branch predictors as in previous schemes. The overhead associated with our scheme is due to the power gating/power up of the ROB, RAT and LSQ units and partial power on/off of fetch, decode and execution units while switching between OOO and modes. Due to the low overhead associated with our morphing scheme, we can dynamically morph from one mode to another at a fine-grained instruction granularity. As mentioned earlier, the InO mode with reduced architectural units provides better energy efficiency at the cost of lower performance. It is critical that we move into InO mode only when we expect increased energy benefits without compromising performance significantly. To minimize the performance loss encountered while running in this dynamic configuration (OOO + InO modes), using ED^2P as the decision metric assigns higher weight to performance than energy.

IV. Conclusions

Applications experience a change in characteristics over time. Hence, different core configurations may be better suited for lower energy and higher performance at different time instances. Traditionally, Asymmetric Multicores (AMP) have been considered to support the diverse needs of applications. In a typical scenario, depending on the current application characteristics, threads are swapped between the available cores in an AMP such that the target objective function related to energy and performance is optimized. Excessive thread migration overheads limit the instruction granularity at which such thread swapping decisions may be made, even though many opportunities present themselves at fine grain granularities. In this paper, we have considered an architecture that is capable of realizing these benefits at finer instruction granularities. In the proposed scheme, depending on the application characteristics, a superscalar OOO processor may morph itself into an in-order (InO) core at runtime, if deemed to be beneficial. Such morphing is already been supported in certain Intel processors, bearing testament to the feasibility and practicality of this approach.

V. Acknowledgement

This research was supported in part by grants 0903191 and 1201834 from the National Research foundation.

References

[1] J. Held *et al.*, "White paper from a few cores to many: A tera-scale computing research review," 2006.

[2] M. Pericas *et al.*, "A flexible heterogeneous multi-core architecture," in *Proceedings of the 16th International Conference fmdahlon Parallel Architecture and Compilation Techniques*, ser. PACT '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 13–24.

[3] R. Kumar *et al.*, "Single-isa heterogeneous multi-core architectures: the potential for processor power reduction," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, dec. 2003.

[4] R. Rodrigues *et al.*, "Performance per watt benefits of dynamic core morphing in asymmetric multicores," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, oct. 2011, pp. 121 –130.

[5] M. Becchi and P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *Proceedings of the 3rd conference on Computing frontiers*, ser. CF '06, 2006.

[6] A. Lukefahr *et al.*, "Composite cores: Pushing heterogeneity into a core," in *International Symposium on Microarchitecture (MICRO), 2012 IEEE/ACM International Symposium on*, dec. 2012.

[7] D. Koufaty *et al.*, "Bias scheduling in heterogeneous multi-core architectures," in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10.

[8] C. Kim *et al.*, "Composable lightweight processors," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 40. Washington, DC, USA: IEEE Computer Society, 2007, pp. 381–394.

[9] D. Tarjan *et al.*, "Federation: Repurposing scalar cores for out-of-order instruction issue," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, june 2008, pp. 772 –775.

[10] Khubaib *et al.*, "Morphcore: An energy-efficient microarchitecture for high performance ilp and high throughput tlp," in *International Symposium on Microarchitecture (MICRO), 2012 IEEE/ACM International Symposium on*, dec. 2012.

[11] G. Contreras and M. Martonosi, "Power prediction for Intel XScale reg; processors using performance monitoring unit events," in *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on*, aug. 2005, pp. 221 – 226.

[12] K. Singh *et al.*, "Real time power estimation and thread scheduling via performance counters," *SIGARCH Comput. Archit. News*, vol. 37, no. 2, pp. 46–55, Jul. 2009.